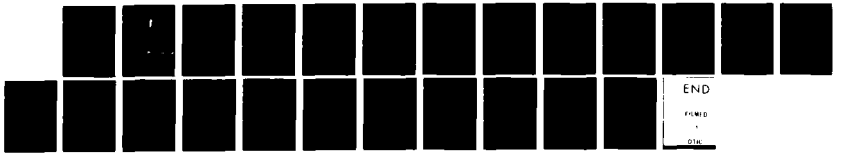
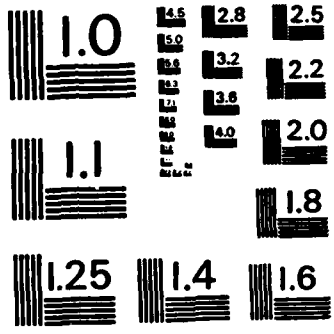
AD-A121 292 PARALLEL ALGORITHMS FOR IMAGE ANALYSIS(U) MARYLAND UNIV 1/1
COLLEGE PARK COMPUTER VISION LAB A ROSENFELD JUN 82
TR-1180 AFOSR-TR-82-0945 AFOSR-77-3271

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

(10)

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 82-0945	2. GOVT ACCESSION NO. AD-A121292	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PARALLEL ALGORITHMS FOR IMAGE ANALYSIS		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL
		6. PERFORMING ORG. REPORT NUMBER TR-1180
7. AUTHOR(s) Azriel Rosenfeld		8. CONTRACT OR GRANT NUMBER(s) AFOSR-77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Vision Laboratory, Computer Science Center, University of Maryland, College Park MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Directorate of Mathematical & Information Sciences Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE June 1982 13. NUMBER OF PAGES 20
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing; image analysis; parallel processing; cellular computers.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper surveys the types of algorithms that are used in image processing and analysis, at both the pixel level and the region level, and discusses how such algorithms might be implemented in parallel using various types of cellular architectures.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 82-0945	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PARALLEL ALGORITHMS FOR IMAGE ANALYSIS		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL
		6. PERFORMING ORG. REPORT NUMBER TR-1180
7. AUTHOR(s) Azriel Rosenfeld		8. CONTRACT OR GRANT NUMBER(s) AFOSR-77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Vision Laboratory, Computer Science Center, University of Maryland, College Park MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Directorate of Mathematical & Information Sciences Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE June 1982 13. NUMBER OF PAGES 20
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Image processing; image analysis; parallel processing; cellular computers.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This paper surveys the types of algorithms that are used in image processing and analysis, at both the pixel level and the region level, and discusses how such algorithms might be implemented in parallel using various types of cellular architectures.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TR-1180
AFOSR-77-3271

June, 1982

PARALLEL ALGORITHMS FOR IMAGE ANALYSIS

Azriel Rosenfeld

Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, MD 20742

ABSTRACT

This paper surveys the types of algorithms that are used in image processing and analysis, at both the pixel level and the region level, and discusses how such algorithms might be implemented in parallel using various types of cellular architectures.

The support of the U.S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Janet Salzman in preparing this paper.

1. Introduction

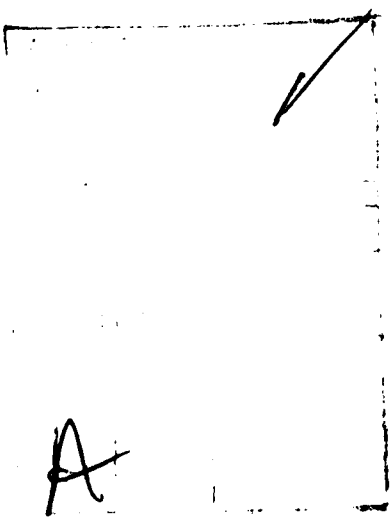
Image processing and analysis (IPA) systems employ a wide variety of techniques for image encoding, transformation, segmentation, and property measurement. Many of these techniques are suitable for efficient parallel implementation. This paper defines some general classes of IPA algorithms, and indicates how such algorithms can be implemented in parallel using various types of "cellular" multiprocessor architectures.

IPA has a large and rapidly growing literature. There are at least a dozen textbooks [1-12] covering major parts of the subject, aside from books on specific topics and collections of papers. An annual bibliography (the most recent is [13]), covering primarily the non-application oriented U.S. literature, currently includes about 1000 references per year. References on specific (classes of) algorithms will not be given in this paper.

It was proposed about 25 years ago [14] that many IPA algorithms could be implemented in parallel using a "cellular array" machine - i.e., a two-dimensional array of processors ("cells"), operating synchronously, each of which can communicate with its neighbors in the array. Several machines of this type, with array sizes of up to 128×128, have actually been constructed. Numerous IPA algorithms suitable for implementation on a cellular array have been developed; for general discussions of this subject see [15,16]. Recently there has been some interest in using

"pyramids" of cellular arrays, of sizes $2^n \times 2^n$, $2^{n-1} \times 2^{n-1}$,..., 2×2, 1×1, where each cell can communicate not only with its neighbors ("brothers") on its own level, but also with its four "sons" on the level below and with its "father" on the level above [15,17].

Cellular arrays or pyramids are suitable for many types of IPA operations at the pixel level. On the other hand, some image analysis operations, involving regions in an image, do not make use of pixel arrays, but rather use other types of data structures to represent regions and their relationships. For such operations, a more general class of graph-structured cellular machines would be appropriate, in which the cells correspond to the nodes of a graph, and can communicate with their neighbors as defined by the arcs of the graph. On such "cellular graph" machines see [18, 19]; on architectures corresponding to more specific types of data structures see [20-22].



2. Pixel-level operations

A digital image is a rectangular array of pixels ("picture elements"). A pixel is usually integer-valued (most commonly, 8-bit integers are used), but it can also be real- or complex-valued, or vector-valued (in the case of color or multispectral imagery). In this section we describe IPA algorithms which operate on pixel arrays.

2.1 Point and local operations

Most of the operations commonly performed in IPA take images into images, where the value of a pixel in the output image depends only on the value(s) of the corresponding pixel, and possible of some of its neighbors, in the input image (or images). Some examples of such operations are:

a) Contrast enhancement by grayscale transformation: the new value of a pixel depends only on its old value, as defined by a given mapping

b) Sharpening by (e.g.) Laplacian filtering, involving the difference between the pixel and the average of its neighbors

c) Smoothing by local averaging (taking an average, possibly weighted, of the pixel and (some of) its neighbors), by median filtering (using the median of the pixel and its neighbors), by averaging of multiple images in register, etc.

d) Segmentation by thresholding (the new value of a pixel is 1 or depending on whether or not the old value exceeds a threshold), or more generally, by classification of the pixels based on a set of property values (color components, local property values, etc.)

e) Edge (or other local feature) detection, based on computing differences between neighboring pixels

f) Expanding or shrinking: the new value of a pixel is the max or min of the vlaues of a set of its neighbors (in some cases, e.g., that of thinning operations, additional conditions must also be satisfied before the value of a pixel is changed)

Note that some of these operations are linear (and hence are convolutions), but most of them are not.

Operations of these types can be performed very efficiently on a cellular array machine in which (ideally) there is a processor associated with each pixel. Each processor collects the values of its neighbors, if necessary, and then computes the required function of these vlaues. The time required to do this depends on the neighborhood size and the complexity of the operation, but not on the size of the image. If there are not enough processors, we can process the image blockwise, using enough overlap between blocks to avoid border effects.

2.2 Transforms

Various types of integral transforms (or their discrete versions) are often performed on images; the Fourier transform is the most common example. Here again the output image is an array of the same size as the input image, but there is no longer a correspondence between their pixels. The transforms are usually separable, so that they can be performed first row-wise, then column-wise; the value of a pixel in the transform is then a linear combination of the values of the pixels in that row (or column) of the image.

When transforms are done on a conventional computer, one can use efficient algorithms (e.g., the "fast Fourier transform") which require $O(n \log n)$ operations, rather than $O(n^2)$, on each row (or column); the total computational cost for an $n \times n$ image is thus $O(n^2 \log n)$ (We ignore here the problem of accessing the image from peripheral storage, and the possible need to transpose the image in order to access it efficiently column-wise as well as row-wise.) On a cellular array machine, the rows (or columns) can be transformed in parallel, and the time required for each row is $O(n)$ (each pixel must be multiplied by n coefficients, and the results must be grouped and summed); thus the overall time is also $O(n)$.

Many useful types of operations (e.g., convolution operations) can be performed on an image by taking its Fourier transform, multiplying the transform pointwise by an appropriate weighting function (or multiplying the transforms of two images pointwise),

and then taking the inverse Fourier transform of the result to obtain the processed image. This too requires only O(n) time on a cellular array machine. For convolutions involving large numbers of weights, this may be more efficient than performing the convolution directly by parallel collection of information from the neighbors of each pixel.

2.3 Geometric operations

Another class of image-to-image operations involves geometric transformations of an image - e.g., rescaling, rotation, or arbitrary "warping" (to correct geometric distortions, or to achieve registration with another image). Here the output pixels do correspond to the input pixels, but not in a simple one-to-one fashion (even a transformation such as rotation, when performed digitally, is not one-to-one). To perform such a transformation on an image, one must compute, for each pixel in the output array, the corresponding positions in the input array (which will not, in general, coincide with the position of an input pixel). One must then assign a value to that output position by interpolation on the nearby input values.

The basic method of performing a geometric transformation on a cellular array machine is to assign an output pixel to each processor, and scan the input image over the array so that each processor eventually sees every input pixel; the processor can thus collect the input pixel values that it needs to compute its output value. If it is actually necessary to scan over the entire input image, this method is not very efficient, since it requires $O(n^2)$ steps; but this is hard to avoid for transformations such as rotation, where the information needed to compute the value of each output pixel comes from a different position, relative to that pixel, in the image. If the needed input information is always in a given range of positions relative to the output pixel, we need only scan over that range, which is much more efficient. Other simplifications are possible for special types of transformations, e.g., for scale changes.

2.4 Property measurement

We now consider operations that map an input image into a (set of) property value(s), rather than into an output image. Examples of such operations include:

a) Determining the presence or absence of a particular pixel value in the image, or computing statistics of the values (min, max, median, range, mean, standard deviation, etc.)

b) Counting the number of occurrences of a particular value - e.g., the number of 1's in a two-valued image gives the area of the set of 1's; the numbers of pixels having each possible value define the gray level histogram of the image; the numbers of pairs of pixels in a given relative position that have each possible pair of values define a gray level "cooccurrence matrix" of the image, which is useful in describing its texture. Note that the last two examples involve sets of k or k^2 properties, where k is the number of gray levels.

c) Counting the number of connected components of pixels having a particular value (this is the standard method of counting objects in a segmented image).

On a cellular array machine, such operations require O(n) time, since the pixel values must be brought togther in one place in order to count them or compute their statistics, and this requires a number of communication steps proportional to the array diameter.

Counting connected components requires a preliminary step in which each component is reduced to a single pixel, but this too can be done in time O(n) using a special type of shrinking process.

Statistics computation and counting can be done in time O(log n) on a cellular pyramid machine (see Section 1). Each pixel passes its value to its "father" on the level above it, which counts or consolidates the values received from its sons and passes on the results to its own father; thus after log n steps (the number of levels), the cell at the apex of the pyramid has the final desired value. Note that this process makes use only of the vertical connections (between levels) in the pyramid, but not of the connections within a given level; thus it requires only a cellular tree machine having the pixels at its leaves. Connected component counting does require horizontal connections in the base of the pyramid in order to carry out the shrinking step, which still takes O(n) time; thus a pyramid provides no great advantage in the case of component counting, and it is also of no great benefit in image-to-image operations.

3. Region-level operations

3.1 Region representations

A region in an image, or in fact any subset of an image, can be represented by a two-valued "overlay" image in which the pixels belonging to the subset have value 1, and all other pixels have value 0. This representation has the advantage of being in registration with the original image, but it has the disadvantage of requiring n^2 bits of memory no matter how simple the given region may be. Regions can be represented in other ways which require less memory for simple regions. Moreover, we can compute properties of regions, and derive new regions from given ones, by operating directly on the representations. Such operations too can be implemented in parallel, but an array of processors is no longer the appropriate architecture, since the representation is no longer array-like. In this section we discuss some standard region representations and the possibility of operating on them in parallel using appropriate multiprocessor architectures.

A region can be defined by specifying its borders (there is more than one border if the region has holes); for each border, this requires the coordinates of a starting point, together with a sequence of codes defining the succession of moves from pixel to pixel around the border (3 bits per move, since successive pixels are neighbors, and a pixel has only 8 immediate

neighbors). A natural architecture for parallel processing of border codes [20] consists of processors connected in ring structures, with each ring representing a border (one code per processor). Computing properties of the regions represented in this way takes time O(m), where m is the border length, as it would on a sequential machine; but certain tasks that take time $O(m^2)$ when done sequentially can be done in O(m) on a ring machine - e.g., computing the border codes of the union or intersection of two regions, given the codes of the regions.

Another way to represent a region is to regard each row of the image that meets the region as a succession of runs (=maximal sequences) of 0's alternating with runs of 1's. Each row is determined by specifying the starting value (1 or 0) and the sequence of run lengths. Region properties, and run length codes of derived regions, can be computed directly from the code(s) of the given region(s). A simple architecture for processing run length codes in parallel might consist of strings of processors, where each string contains the run lengths for a given row. Greater efficiency could be achieved by allowing direct connections between strings representing adjacent rows, with the processor representing a given run connected directly to the processors representing runs on the adjacent rows that overlap the given run. This approach to parallel region processing does not seem to have been systematically investigated (but see [21]).

Runs are maximal horizontal "strips" of constant-value pixels; a more compact way of representing a region is to use maximal two-dimensional blocks of constant-value pixels. Each such block is defined by specifying its center and radius, and the region is then the union of the blocks. A representation of this type, known as the medial axis transformation, was introduced about 20 years ago; but it has not been used extensively for region processing, because it is difficult to compute region properties or to derive new regions from it directly, due to the fact that the blocks overlap one another and are not organized in a systematic way. In some cases, it may be possible to represent a region as a union of "generalized ribbons", where each ribbon is a union of maximal blocks whose centers all lie on a curve. Such a representation would be much more manageable, and could be processed in parallel by assigning the code of each curve (i.e., the sequence of moves and the corresponding radii) to a string of processors; this possibility has not been investigated.

Another type of maximal-block region representation can be constructed by recursively subdividing the given two-valued image into quadrants, subquadrants, ... until blocks of constant value are reached. The resulting block structure can be represented by a tree of degree 4 (a quadtree) in which the root corresponds to the entire image, and the sons of a node correspond to its quadrants. For an n×n image, where n is a

power of 2, the height of the tree is at most $\log_2 n$, and each leaf of the tree represents a block of the image consisting entirely of 0's or 1's. Region properties, and quadtree representations of derived regions, can be computed from the quadtree(s) of the given region(s) sequentially by traversing the tree(s). Quadtree-connected sets of processors can be used to perform many of these operations in parallel very efficiently [22]. A generalization of this approach can be used to represent a multivalued image as a union of homogeneous blocks (e.g., blocks of constant value, or blocks in which the standard deviation of pixel values is low), where we divide a block into quadrants iff it is nonhomogeneous.

3.2 Region properties and relations

The region representations described above are especially useful in manipulating data bases of regions, e.g., in digital cartography. In image analysis, such representations are used for measuring region properties and for deriving new regions from given ones. In this section we consider a more abstract level of processing in which regions are not completely specified, but are represented by lists of their properties. An image segmentation can be represented, at this level, by a graph structure in which the nodes correspond to regions, labeled with lists of property values; and the arcs correspond to related pairs of regions (e.g., adjacent), labeled with relation values (e.g., length of common border).

A segmentation can be modified by merging pairs of regions based on information provided by the graph representation, without any need to refer to the original image; and the graph of the new segmentation can be constructed directly from that of the given segmentation. For example, suppose the graph contains information about the area, perimeter, and average pixel value of each region, and the length of common border of each adjacent pair of regions. The following are some possible criteria for merging a pair of adjacent regions: their averages values are very similar; their areas are very different; their length of common border is a large fraction of (one of) their perimeters. Their criteria can be checked directly from the graph. Moreover, if we decide to merge two regions, we can construct the new graph directly from the old

one, by replacing the two old nodes with a single node connected to all of the old nodes' other neighbors. The properties of the new node can be computed as follows: its area is the sum of the old nodes' areas; its average pixel value is the weighted average of the old nodes' averages, weighted by their areas; its perimeter is the sum of the old nodes' perimeters minus the length of their common border. Finally, the lengths of common border between the new node and its neighbors can be computed immediately from these lengths for the old nodes; they remain the same except in the case of a neighbor common to both of the old nodes, where the two lengths must be added.

Region merging processes such as that just described can be carried out in parallel using a network of processors in which a processor is assigned to each region, and the processors corresponding to adjacent regions can communicate directly - in other words, the processor network is isomorphic to the region adjacency graph. Thus each processor can examine the information stored at its neighbors (and at their arcs) and decide whether merging is possible. It should be pointed out that when merging is done in parallel, the decision to merge a pair of regions must be agreed to by both of them; if a region were allowed to make such a decision on its own, we might find that region A merges with region B and at the same time B merges with C, leading to an inconsistency (there are new nodes representing A+B and B+C, but no node for A+B+C, which in any case may not be an appropriate merge). To avoid this, only disjoint pairs should be allowed to merge. For further discussion of parallel region-level processing see [18].

4. Concluding remarks

This paper has reviewed some of the basic types of operations used in image processing and analysis, at both the pixel and region levels, and has described idealized multiprocessor configurations suitable for carrying out such operations in parallel.

We have seen that for pixel-level operations taking images into images, a cellular array architecture, with processors connected in a regular grid, is very natural. For image property measurement, on the other hand, greater efficiency can be achieved by using tree-structured connections, with the processors at the leaves of the tree. For parallel processing of regions defined by border codes, ring-connected processors are appropriate. Other connection schemes are suitable if the regions are defined by maximal blocks, e.g., by run length codes or by quadtrees. At a more abstract level, when regions are represented by lists of properties, region merging can be carried out in parallel using a network of processors connected in the same way as the region adjacency graph.

Parallel region-level processing generally requires a much smaller number of processors than parallel processing at the pixel level. A cellular array machine for parallel processing of a 512×512-pixel image, one processor per pixel, requires $\frac{1}{4}$ million processors, which is not yet practical; but a region-level processor might require only a few hundred processors per region (depending on their complexity), or even fewer processors to handle a region adjacency graph (depending on the

complexity of the segmentation). These numbers of processors are quite manageable, but their interconnections pose a problem. For pixel-level processing, the images to be processed will all be of the same size, and the neighbor interconnections are the same for every image, so that a cellular array machine can be hard-wired once and for all. For processing at the region level, on the other hand, the interconnections vary from image to image, since the shapes of the regions cannot be predicted in advance. Worse yet, we may even want the interconnections to vary in the course of a computation, as new regions are defined or old regions merged. This calls for some type of reconfigurable multiprocessor architecture [19], where ideally the reconfiguration itself should take place in parallel. For some types of representations (e.g., border codes, for which linked rings of processors can be used), such reconfiguration may be relatively easy; but for other representations, requiring tree or graph interconnections, parallel reconfiguration may not be easy to realize in such a way as to avoid serious interprocessor communication bottlenecks. As advances in hardware technology make it possible to build large multiprocessor networks, the problems involved in designing efficient systems for parallel image processing and analysis, both at the pixel and region levels, will have to be addressed.

References

1. H. C. Andrews, Computer Techniques in Image Processing, Academic Press, NY, 1970.

2. D. Ballard and C. Brown, Computer Vision, Prentice-Hall, Englewood Cliffs, NJ, 1982.

3. K. R. Castleman, Digital Image Processing, Prentice-Hall, Englewood Cliffs, NJ, 1979.

4. R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis, Wiley, NY, 1973.

5. R. C. Gonzalez and P. Wintz, Digital Image Processing, Addison-Wesley, Reading, MA, 1977.

6. E. L. Hall, Computer Image Processing and Recognition, Academic Press, NY, 1979.

7. T. Pavlidis, Structural Pattern Recognition, Springer, NY, 1977.

8. T. Pavlidis, Algorithms for Graphics and Image Processing, Computer Science Press, Rockville, MD, 1982.

9. W. K. Pratt, Digital Image Processing, Wiley, NY, 1978.

10. A. Rosenfeld, Picture Processing by Computer, Academic Press, NY, 1969.

11. A. Rosenfeld and A. C. Kak, Digital Picture Processing, Academic Press, NY, 1976; second edition (two volumes), 1982.

12. J. Serra, Image Analysis and Mathematical Morphology, Academic Press, 1982.

13. A. Rosenfeld, Picture Processing: 1981, Computer Graphics Image Processing 19, 1982, 35-75.

14. S. H. Unger, A computer oriented toward spatial problems, Proc. IRE 46, 1958, 1744-1750.

15. A. Rosenfeld, Picture Languages, Academic Press, NY, 1979.

16. C. R. Dyer and A. Rosenfeld, Image processing by memory-augmented cellular automata, IEEE Trans. PAMI-3, 1981, 29-41.

17. C. R. Dyer and A. Rosenfeld, Triangle cellular automata, Info. Control 48, 1981, 54-69.

18. A. Rosenfeld and A. Wu, Parallel computers for region-level image analysis, Pattern Recognition 15, 1982, 41-50.

19. A. Rosenfeld and A. Wu, Reconfigurable cellular computers, Info. Control, in press.

20. T. Dubitzki, A. Wu, and A. Rosenfeld, Parallel computation of contour properties, IEEE Trans. PAMI-3, 1981, 331-337.

21. J. S. Todhunter and C. C. Li, A new model for parallel processing of serial images, Proc. 5th Intl. Conf. on Pattern Recognition, 1980, 493-496.

22. T. Dubitzki, A. Wu, and A. Rosenfeld, Region property computation by active quadtree networks, IEEE Trans. PAMI-3, 1981, 626-633.